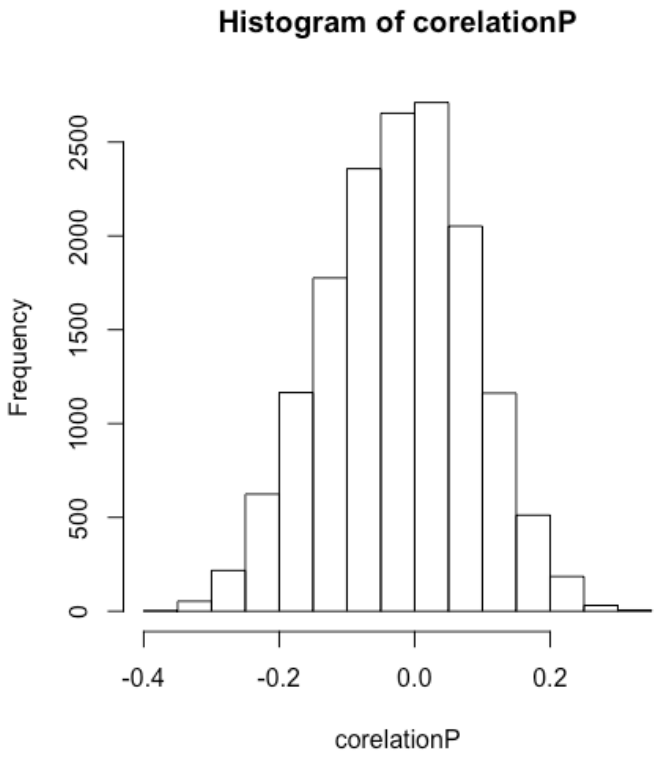
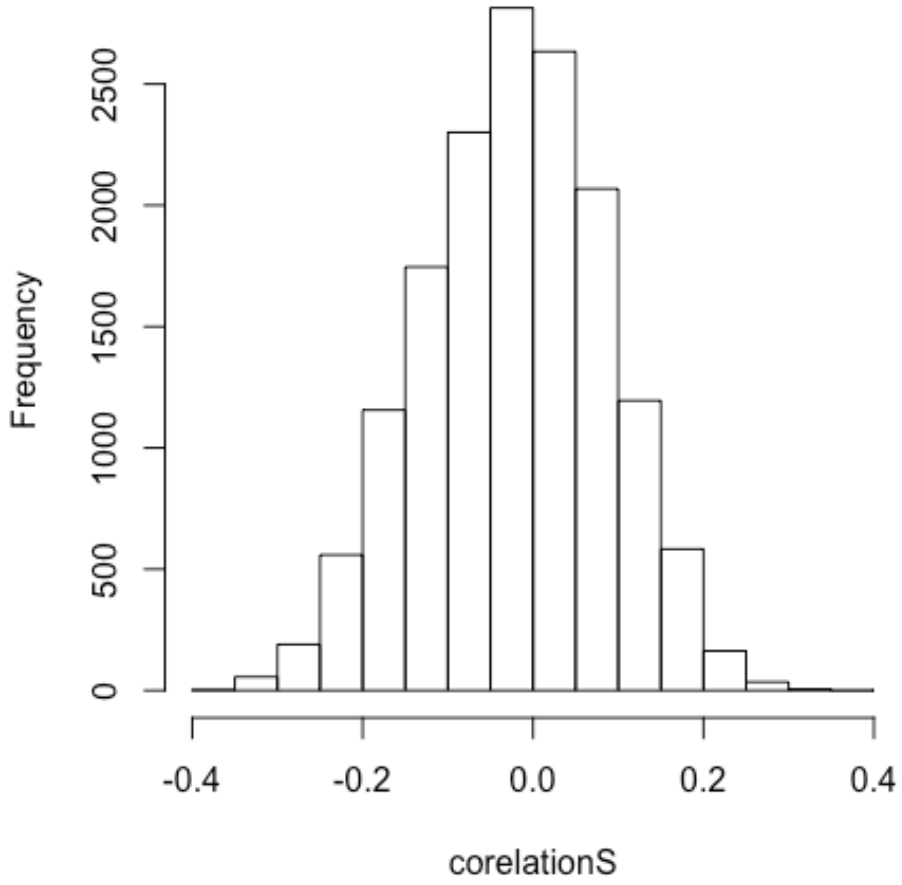
```
install.packages("rmarkdown")
library(rmarkdown)

load("~/Desktop/GlioblastomaWide.rda")#tu podaj ścieżkę do swojego pliku .rda
data <- GlioblastomaWide

data_kaggle <- read.table(file = "~/Desktop/GlioblastomaWide.csv", sep=",", dec=".", header =
TRUE)
data_test_kaggle <- read.table(file = "~/Desktop/final.csv", sep=",", dec=".", header = TRUE)

#data_kaggle <- read.table(file = "https://inclass.kaggle.com/c/glioma-survival-predictions/
download/GlioblastomaWide.csv", sep=",", dec=".", header = TRUE)
#data_test_kaggle <- read.table(file = "https://inclass.kaggle.com/c/glioma-survival-predictions/
download/final.csv", sep=",", dec=".", header = TRUE)

# Celem projektu było znalezienie reguły decyzyjnej, która miałaby pomóc w ocenie, czy dany
pacjent z nowotworem Glioblastoma przeżyje najbliższy rok.
# Na podstawie otrzymanych wyników nie potrafiliśmy ocenić dalszego przeciętnego trwania życia
pacjenta.
#
#
# W naszych badaniach postanowiliśmy przyjąć, że pacjenci powyżej {wpisz właściwy wiek} roku
życia mogą umrzeć z zupełnie innego powodu niż ten nowotwór.
# W wyborze tego wieku kierowaliśmy się dostępnymi tablicami demograficznymi z przeciętnym
dalszym trwaniem życia. Uwzględnialiśmy natężenie wymierania w badanej populacji.


#-------------------------------------------------------------------------------
#---------------------------------- etap 1 -------------------------------------
#-------------------------------------------------------------------------------

#-------------------------przygotowanie danych-----------------------------------

data$Cluster<-factor(data$Cluster)
data$death1y <- 1-(as.numeric(as.factor(data$death1y)) - 1)

for(i in 1:16115){
  if(mean(is.na(data[,16120-i])) > 0.2){
    data[,16120-i] <- NULL
  } else {
    sr <- mean(na.omit(data[,16120 - i]))
    for(j in 1:125){
      if(is.na(data[j, 16120 - i])){
        data[j, 16120 - i] <- sr
      }
    }
  }
}

#dane o wieku

# W ten sposób przedstawia się dystrybuanta empiryczna rozkładu wieku pacjenta, u którego
zdiagnozowano nowotwór. Można na jej podstawie stwierdzić tylko przeciętny wiek, w którym
# ludzie zapadają na nowotwór Glioblastoma.
```

```
summary_age<-summary(data$age)
boxplot(data$age, col="grey", horizontal = TRUE)
hist(data$age, col = "grey")
plot(ecdf(data$age), las=1)

#dzielimy dane na dwa zbiory, alive i dead
alive <- data[data$death1y == 1,]
dead <- data[data$death1y == 0,]

# Posłużyliśmy się testem korelacji Pearsona i testem korelacji Spearmana w celu zbadania, czy istnieje jakiś wpływ ekspresji genów na możliwość przeżycia przez pacjenta pierwszego roku od diagnozy.
# Zabrakło w otrzymanej tabeli danych o momencie zachorowania pacjenta. Nie wiemy, jak długo chorował pacjent przed zdiagnozowaniem.

#definiuje funkcje korelacji
corelation_function_spearman <- function(arg){
  return(cor(data$death1y, arg, use = "complete.obs",method ="spearman"))
}

#definiuje funkcje korelacji
corelation_function_pearson <- function(arg){
  return(cor(data$death1y, arg, use = "complete.obs",method ="pearson"))
}
```






```
#mapuje funkcje korelacji na danych
corelationS <- sapply(data[,5:ncol(data)], FUN=corelation_function_spearman)

corelationP <- sapply(data[,5:ncol(data)], FUN=corelation_function_pearson)

hist(corelationS)
hist(corelationP)

# Na podstawie wyżej wspomnianych testów wybraliśmy 30 najbardziej istotnych genów. Poniżej przedstawiamy ich listę.
# wypisuje 20 genow o najwyzszych wartosciach
topS<-names(head(sort(abs(corelationS),decreasing = TRUE),30))
topP<-names(head(sort(abs(corelationP),decreasing = TRUE),30))

plot(data[,topS[1:10]],col=c("red","green")[as.factor(data$death1y)], pch=18)

# Użyliśmy też testu t-Studenta, przy którym dzieliliśmy ludzi na tych, którzy przeżyli rok i nie przeżyli roku. Rozpatrywaliśmy wynik tego testu dla każdego genu z osobna.
# Dokonaliśmy poprawki FDR i poprawki Bonferroniego p-wartości testów. p-wartości dla poprawki Bonferroniego są skupione w jedynce, a te FDR są bardziej jednostajnie rozłożone na odcinku (0,1).
# test t-studenda dla kazdej kolumny dla alive i dead
studentsRes<-NULL
cols <- ncol(data)
for(i in 5:cols){
  a<-alive[,i]
```

```
  d<-dead[,i]
  t<-t.test(d, a)$p.value
  studentsRes[(i-4)] <- t
}
pvalue_poprawkaFDR<-p.adjust(studentsRes, method = "fdr")
pvalue_poprawkaBONF<-p.adjust(studentsRes, method = "bonferroni")
rpvalue_poprawkaFDR<-as.data.frame(pvalue_poprawkaFDR)
rpvalue_poprawkaBONF<-as.data.frame(pvalue_poprawkaBONF)
rpvalue<-as.data.frame(studentsRes)

# Poniżej przedstawiamy: histogram otrzymanych wyników testu t-Studenta oraz kolejno
histogramy przyjmowanych p-wartości przy poprawce FDR i poprawce Bonferroniego.

library(ggplot2)
ggplot(data=rpvalue, aes(studentsRes)) +
geom_histogram(aes(fill=..count..),col="black",breaks=seq(0,1,0.025))+
scale_fill_gradient("Count", low = "green", high = "red")
ggplot(data=rpvalue_poprawkaFDR, aes(pvalue_poprawkaFDR)) +
geom_histogram(aes(fill=..count..),col="black",breaks=seq(0,1,0.025))+
scale_fill_gradient("Count", low = "green", high = "red")
ggplot(data=rpvalue_poprawkaBONF, aes(pvalue_poprawkaBONF)) +
geom_histogram(aes(fill=..count..),col="black",breaks=seq(0,1,0.025))+
scale_fill_gradient("Count", low = "green", high = "red")

#-------------------------------------------------------------------------------
#---------------------------------- etap 2 -------------------------------------
#-------------------------------------------------------------------------------
# Aby ocenić, czy dany pacjent przeżyje rok posłużyliśmy się metodami: regresji logistycznej,
Naive Bayes, Random Forest, drzewami klasyfikacyjnymi i Support Vector Machine.
# W tym celu podzieliliśmy zbiór danych zbiór testowy i zbiór uczący.

# zbior testowy i treningowy

#install.packages("caret", dependencies = c("Depends"))
library("caret")
set.seed(1313)


data$death1y <- as.factor(data$death1y)

inds <- createDataPartition(data$death1y, p = 0.75)
train <- data[inds[[1]],]
test  <- data[-inds[[1]],]
wybrane <- c("death1y",  topS)

# -------------------------regresja logistyczna probit-----------------------------------

# Na naszych danych algorytm scorów Fishera działa bardzo dobrze (używa bardzo małej liczby
iteracji). Nie wnosi to jednak nic do rozwiązania naszego problemu.
# Nasz model ma AIC=116.58. Porównamy go potem z innymi. Ten, który będzie miał najmniejszą
wartość tego kryterium będzie modelem najlepszym do zadanej nam predykcji.


modelProbit <- glm(death1y ~ ., data=train[,wybrane],family=binomial(link="probit"))
poprawionyProbit <- step(modelProbit, method="backward")
summary(modelProbit)
```

```
foldy <- createFolds(data$death1y, k = 10)
errorsProbit <- lapply(foldy, function(ind) {
  model<-step(glm(death1y ~ .,
data=train[,wybrane],family=binomial(link="probit")),method="backward")
  predykcja1 <- predict(model, data[ind,wybrane],type="response")
  mean(abs(predykcja1-as.numeric(data[ind, "death1y"])))
})
mean(as.numeric(errorsProbit))
hist(unlist(errorsProbit), col="grey")

# ----------regresja logisyczna dla dobranego zbioru testowego i treningowego -----

# W tym przypadku AIC pokazuje wartość 117.22, co świadczy, że to podejście jest gorsze od
regresji logistycznej probit.
# Mniejsza liczba iteracji świadczy wyłącznie o dobrej zbieżności tej metody.

model <- glm(death1y ~ ., data=train[,wybrane],family = binomial(logit))
summary(model)
poprawiony <- step(model, method="backward")


# dla k fold k = 10

foldy <- createFolds(data$death1y, k = 10)
errors <- lapply(foldy, function(ind) {
  model<-step(glm(death1y~.,data[-ind,wybrane],family="binomial"),method="backward")
  predykcja1 <- predict(model, data[ind,wybrane],type="response")
  mean(abs(predykcja1-as.numeric(data[ind, "death1y"])))
})
mean(as.numeric(errors))
hist(unlist(errors), col="grey")

# rysowanie krzywej ROCR

library("plotROC")
predictionRegr<-predict(poprawiony, test)
#tabela kontyngencji
table(pred=predictionRegr, true=test$death1y)

# Krzywa ROC jest bardziej przesunięta w lewo, co świadczy o tym, że nasz test dokonuje całkiem
skutecznej klasyfikacji.

library("ROCR")

predRegr <- prediction( predictionRegr, test$death1y)
perfRegr <- performance(predRegr,"tpr","fpr")
plot(perfRegr,col="blue")
abline(0,1)

library("OptimalCutpoints")
pref_df_regr <- data.frame(pred = predictionRegr, truth = test$death1y)
oc <- optimal.cutpoints(X = "pred", status = "truth", methods="Youden", data=pref_df_regr,
tag.healthy = "0")
summary(oc)
plot(oc, which=1)
```

```
# ------------------------------drzewo klasyfikacyjne rpart------------------------------
# Posłużyliśmy się również metodą drzewa klasyfikacyjnego rpart. Szkic drzewa klasyfikacyjnego
przedstawia najlepiej, jaka reguła predykcyjna została tutaj opracowana.

library("rpart")
library("rpart.plot")

# model dla wybranego zbioru testowego

drzewo <- rpart(as.factor(death1y) ~ . , data=train[,wybrane], method="class",
control=rpart.control(maxdepth = 10))
rpart.plot(drzewo)

# rysowanie krzywej ROCR

predictionRpartTree<-predict(drzewo, test)[,2]
#tabela kontyngencji
table(pred=predictionRpartTree, true=test$death1y)

# Krzywa ROC położona jest bliżej diagonali, zatem drzewa klasyfikacyjne łatwiej dokonują
błędnego zakwalikowania pacjentów niż metoda glm.
#krzywa ROC
predRpartTree <- prediction( predictionRpartTree, test$death1y)
perfRpartTree <- performance(predRpartTree,"tpr","fpr")
plot(perfRpartTree,col="blue")
abline(0,1)

#optimal cutpoints
pref_df_RpartTree <- data.frame(pred = predictionRpartTree, truth = test$death1y)
oc <- optimal.cutpoints(X = "pred", status = "truth", methods="Youden", data=pref_df_RpartTree,
tag.healthy = "1")
summary(oc)
plot(oc, which=1)

# k fold dla k=10

foldy <- createFolds(data$death1y, k = 10)
errorsRTREE <- lapply(foldy, function(ind) {
  drzewo <- rpart(death1y ~ . , data=data[-ind,wybrane], method="class",
control=rpart.control(maxdepth = 10))
  predykcja1 <- predict(drzewo, newData=data[ind,wybrane])
  mean(abs(predykcja1-as.numeric(data[ind, "death1y"])))
})
mean(as.numeric(errorsRTREE))
hist(unlist(errorsRTREE), col="grey")


# ---------------------------------drzewo klasyfikacyjne ctree--------------------------------


# Poniżej przedstawiamy metody użycia metody drzewa klasyfikacyjnego ctree.
# Otrzymane drzewo ctree jest {moim zdaniem} zbyt płytkie. Wygląda tak, jakby wyraźnie
wskazywało istotny wpływ dwóch genów: CLEC5A i NRG1.
# Otrzymana krzywa ROC jest podobna do tej z drzewa klasyfikacyjnego cpart. Zatem jak na razie
uogólniony model liniowy wychodzi najlepiej.
```

```
install.packages("party")
library("party")
nextTree <- ctree(as.factor(death1y) ~ ., data=train[,wybrane],
controls=ctree_control(maxdepth=10, testtype = "Bonferroni", mincriterion = 0.75))
plot(nextTree)

# rysowanie krzywej ROCR

predictionTree<-predict(nextTree, test)
#tabela kontyngencji
table(pred=predictionTree, true=test$death1y)

#krzywa ROC
predTree <- prediction( as.numeric(predictionTree), test$death1y)
perfTree <- performance(predTree,"tpr","fpr")
plot(perfTree,col="blue")
abline(0,1)

# k fold dla k=10

foldy <- createFolds(data$death1y, k = 10)
errorsCTREE <- lapply(foldy, function(ind) {
  drzewo <- ctree(as.factor(death1y) ~ ., data=train[-ind,wybrane],
controls=ctree_control(maxdepth=10, testtype = "Bonferroni", mincriterion = 0.75))
  predykcja <- predict(drzewo, data[ind,wybrane],type="response")
  mean(abs(as.numeric(predykcja) - as.numeric(data[ind, "death1y"])))
})
mean(as.numeric(errorsCTREE))
hist(unlist(errorsCTREE), col="grey")

# -------------------------------- naive bayes ----------------------------------------

# Kolejną metodą, którą posłużyliśmy się był Naive Bayes.
# Krzywa ROC w przypadku Naive Bayes jest trochę inna niż przy drzewach klasyfikacyjnych, ale
wciąż skupiona w sąsiedztwie diagonali. Wciąż najlepszym sposobem według tego kryterium jest
glm.


install.packages("klaR")
install.packages("e1071")
library("e1071")
library("klaR")
tempData <- data
for(i in 1:15513){tempData[,i] <- factor(tempData[,i])}
indsTemp <- createDataPartition(tempData$death1y, p = 0.75)
tempTrain <- tempData[inds[[1]],]
tempTest  <- tempData[-inds[[1]],]

nb <- NaiveBayes(death1y~.,tempTrain[,wybrane])
partimat(death1y~CTSB+NRG1,data=tempTrain[,wybrane],method="naiveBayes",col.correct =
"black", col.wrong = "red",pch=15,image.colors=c("grey","blue1"))

# rysowanie krzywej ROCR
```

```
predictionNB<- as.data.frame(predict(nb, tempTest,threshold = 0.001,eps=0,type = c("class",
"raw")))[,3]
#tabela kontyngencji
table(pred=predictionNB, true=test$death1y)

#krzywa ROC
predNB <- prediction( predictionNB, test$death1y)
perfNB <- performance(predNB,"tpr","fpr")
plot(perfNB,col="blue")
abline(0,1)

# k fold dla k=10
foldy <- createFolds(tempData$death1y, k = 10)
errorsNB <- lapply(foldy, function(ind) {
  nb <- NaiveBayes(death1y~.,tempData[-ind,wybrane])
  predykcja1 <- predict(nb, tempData[ind,wybrane],type="response")
  mean(abs(as.numeric(predykcja1)-as.numeric(tempData[ind, "death1y"])))
})#tu cos sie sypie
mean(as.numeric(errorsNB))
hist(unlist(errorsNB), col="grey")

# ------------------------------------------random forest----------------------------
# Użyliśmy do predykcji również metody Random Forest. Z otrzymanych wyników w varImpPlot
możemy wnioskować o skuteczności cross-validacji out-of-bad po nauczeniu, ale przed wyborem
próbki bootstrapowej.
# Krzywa ROC pokazała, że metoda Random Forest dokonuje skuteczniejszej klasyfikacji niż
drzewa klasyfikacyjne, ale nie lepszej niż uogólniony model liniowy.

library("randomForest")
randomForestData<-
randomForest(as.factor(death1y)~.,data=train[,wybrane],importance=TRUE,proximity=TRUE)

library("plotROC")
predictionRF<-predict(randomForestData, test)
roc.estimate <- calculate_roc(predictionRF, test$death1y)

#tabela kontyngencji
table(pred=predictionRF, true=test$death1y)

varImpPlot(randomForestData)
importance(randomForestData)

library("ROCR")

pred <- prediction( as.numeric(predictionRF), test$death1y)
perf <- performance(pred,"tpr","fpr")
plot(perf,col="blue")
abline(0,1)

library("OptimalCutpoints")
pref_df <- data.frame(pred = predictionRF, truth = test$death1y)
oc <- optimal.cutpoints(X = "pred", status = "truth", methods="Youden", data=pref_df, tag.healthy =
"0")
summary(oc)
plot(oc, which=1)
```

```
# k fold dla k=10
foldy <- createFolds(tempData$death1y, k = 10)
errorsRF <- lapply(foldy, function(ind) {
  randomForestData<-randomForest(death1y~.,data=train[-
ind,wybrane],importance=TRUE,proximity=TRUE)
  predykcja1 <- predict(randomForestData, data[ind,wybrane],type="response")
  mean(abs(as.numeric(predykcja1)-as.numeric(tempData[ind, "death1y"])))
})
mean(as.numeric(errorsRF))
hist(unlist(errorsRF), col="grey")

#------------------------------------ support vector machine ----------------------

svmModel <- svm(death1y~.,data=train[,wybrane])
summary(svmModel)
predictionSVM<-predict(svmModel, test)
predSVM <- prediction( as.numeric(predictionSVM), test$death1y)
perfSVM <- performance(predSVM,"tpr","fpr")
plot(perfSVM,col="blue")
abline(0,1)

#tabela kontyngencji
table(pred=predictionSVM, true=test$death1y)

# k fold dla k=10
foldy <- createFolds(tempData$death1y, k = 10)
errorsSVM <- lapply(foldy, function(ind) {
  svmModel <- svm(death1y~.,data=train[-ind,wybrane])
  predykcja1 <- predict(svmModel, data[ind,wybrane],type="response")
  mean(abs(as.numeric(predykcja1)-as.numeric(tempData[ind, "death1y"])))
})
mean(as.numeric(errorsSVM))
hist(unlist(errorsSVM), col="grey")



#-------------------------------------------------------------------------------
#---------------------------------- etap 3 -------------------------------------
#-------------------------------------------------------------------------------
# klasyfikatory oparte o stacking

# random forest + logistic regression + svm

#randomForestData<-
randomForest(death1y~.,data=train[,wybrane],importance=TRUE,proximity=TRUE)
#predictionRandomForest<-predict(randomForestData, newdata=test)
#errRF <- sqrt((sum(test$death1y-predictionRandomForest)^2)/nrow(test))

#modelLogisticRegr <- glm(death1y ~ ., data=train[,wybrane],family="binomial")
#poprawionyLR <- step(modelLogisticRegr, method="backward")
#predictionLR<-predict(poprawionyLR, test[,wybrane])
#errLR <- sqrt((sum(test$death1y-predictionLR)^2)/nrow(test))

#svmFit <- svm(death1y~.,data=train[,wybrane])
#svm_predict <- predict(svmFit, newdata=test)
#errSvm <- sqrt((sum(test$death1y-svm_predict)^2)/nrow(test))
```

```
#wybralem sobie 3 modele i dowolnie je sumuje, nie wiem czy to na tym polega

#preds <- (predictionRandomForest + predictionLR) / 2
#errSum <- sqrt((sum(test$death1y - preds)^2)/nrow(test))

#boosting with adabag- tutaj cos sie sypie
install.packages("adabag")
library("adabag")
library("mlbench")
adaboost <- boosting(death1y ~., data=train[,wybrane], boos=TRUE, mfinal=10, control = (minsplit
= 0))
summary(adaboost)

adaboost$trees
adaboost$weights
adaboost$importance
predict(adaboost,test)
t1<-adaboost$trees[[1]]
rpart.plot(t1)

resAda <- predict(adaboost, newdata=data_test_kaggle)
predictionADA<-predict(adaboost, test)
predADA <- prediction( as.numeric(predictionADA), test$death1y)
perfADA <- performance(predADA,"tpr","fpr")
plot(perfADA,col="blue")
abline(0,1)
t2<-adaboost$trees[[2]]
rpart.plot(t2)

t3<-adaboost$trees[[3]]
rpart.plot(t3)
#boosting with xgboost
library("xgboost")
y = data$death1y == "1"
X = model.matrix(death1y~.,data)
gb <- xgboost(X,y,objective="binary:logistic", nrounds=2, max.depth=2)
gb

#laczenie modeli
library(ElemStatLearn)
library(randomForest)
library(caret)

set.seed(33833)
train$death1y <- as.factor(train$death1y)
test$death1y <- as.factor(test$death1y)

fit1 <- train(death1y ~., data=train[,wybrane], method='rf')
fit2 <- train(death1y ~., data=train[,wybrane], method='gbm')

results1 <- predict(fit1, newdata=test[,wybrane])
results2 <- predict(fit2, newdata=test[,wybrane])

combo <- data.frame(results1, results2, y = test$death1y)
fit3 <- train(y ~ ., data = combo, method = "rf")
results3 <- predict(fit3, newdata = test[,wybrane])
```

```
c1 <- confusionMatrix(results1, test$death1y)
c2 <- confusionMatrix(results2, test$death1y)
c3 <- confusionMatrix(results3, combo$death1y)

result_final <- predict(fit1, data_test_kaggle[,wybrane[-1]])

caretStack(models, method="glm")
library(gbm)

library(AppliedPredictiveModeling)

set.seed(62433)

# Train using 3 different models.
fit1 <- train(death1y ~., data=train[,wybrane], method='rf')
fit2 <- train(death1y ~., data=train[,wybrane], method='gbm')
fit3 <- train(death1y ~., data=train[,wybrane], method='lda')

# Run models on testing data.
results1 <- predict(fit1, newdata=test)
results2 <- predict(fit2, newdata=test)
results3 <- predict(fit3, newdata=test)

# Stack models together and combine with random forests.
combo <- data.frame(results1, results2, results3, death1y = test$death1y)
fit4 <- train(death1y ~ ., data = combo, method = "rf")

result_final_2 <- predict(fit4, newdata = data_test_kaggle)

# Run stacked model on testing data.
results4 <- predict(fit4, newdata = test)

# random forests
c1 <- confusionMatrix(results1, test$death1y)
# boosting
c2 <- confusionMatrix(results2, test$death1y)
# lda
c3 <- confusionMatrix(results3, test$death1y)
# Stacked models
c4 <- confusionMatrix(results4, combo$diagnosis)

#------------------------ stacking ver 2 -------------------------------------
library(mlbench)
library(caret)
library(caretEnsemble)
# create submodels
data$death1y[data$death1y=="0"]<-"dead"
data$death1y[data$death1y=="1"]<-"alive"

for(i in 1:16116){
  if(mean(is.na(data_test_kaggle[,16118-i])) > 0.2){
    data_test_kaggle[,16118-i] <- NULL
  } else {
    sr <- mean(na.omit(data_test_kaggle[,16118 - i]))
    for(j in 1:250){
```

```
      if(is.na(data_test_kaggle[j, 16118 - i])){
        data_test_kaggle[j, 16118 - i] <- sr
      }
    }
  }
}

#Linear Discriminate Analysis (LDA)
#Classification and Regression Trees (CART)
#Logistic Regression (via Generalized Linear Model or GLM)
#k-Nearest Neighbors (kNN)
#Support Vector Machine with a Radial Basis Kernel Function (SVM)
#Generalized Boosted Regression Modeling

control <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE,
classProbs=TRUE)
algorithmList <- c('lda', 'rpart', 'glm', 'knn', 'svmRadial', 'gbm', 'rf')

seed <- 7
set.seed(seed)


models <- caretList(as.factor(death1y)~., data=data[,wybrane], trControl=control,
methodList=algorithmList)
results <- resamples(models)
summary(results)
dotplot(results)


#Models: lda, rpart, glm, knn, svmRadial, gbm, rf
#Number of resamples: 30

#Accuracy
#Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
#lda       0.5000  0.6667 0.7500 0.7238  0.7692 0.9167    0
#rpart     0.3333  0.5096 0.5833 0.6045  0.6859 1.0000    0
#glm       0.3846  0.5833 0.7212 0.6975  0.7692 1.0000    0
#knn       0.5000  0.6282 0.6923 0.6955  0.7692 0.8571    0
#svmRadial 0.4615  0.6282 0.7500 0.7155  0.7816 0.9167    0
#gbm       0.5385  0.6667 0.7500 0.7474  0.8462 0.9231    0
#rf        0.4167  0.6667 0.7500 0.7418  0.8429 1.0000    0

#Kappa
#Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
#lda        0.00000 0.33330 0.5000 0.4484  0.5412 0.8333    0
#rpart     -0.33330 0.01786 0.1667 0.2102  0.3690 1.0000    0
#glm       -0.20930 0.16670 0.4405 0.3952  0.5384 1.0000    0
#knn        0.00000 0.24600 0.3734 0.3900  0.5412 0.7143    0
#svmRadial -0.07059 0.25980 0.5000 0.4304  0.5639 0.8333    0
#gbm        0.07143 0.33330 0.5000 0.4939  0.6905 0.8471    0
#rf        -0.16670 0.33330 0.5000 0.4842  0.6845 1.0000    0

# correlation between results
modelCor(results)
splom(results)
```

```
#modele powinny miec niska korelacje pomiedzy soba, kiedy wykorzystujemy stacking, >0.75 to zle

#            lda       rpart       glm       knn     svmRadial   gbm        rf
#lda        1.0000000 0.4731959 0.7796056 0.5449545 0.3988154 0.3805589 0.4157637
#rpart      0.4731959 1.0000000 0.3628292 0.3693462 0.4375850 0.4963381 0.5671968
#glm        0.7796056 0.3628292 1.0000000 0.3849873 0.3473833 0.4984669 0.3813019
#knn        0.5449545 0.3693462 0.3849873 1.0000000 0.3786169 0.3937915 0.4517406
#svmRadial 0.3988154 0.4375850 0.3473833 0.3786169 1.0000000 0.6196128 0.7723533
#gbm        0.3805589 0.4963381 0.4984669 0.3937915 0.6196128 1.0000000 0.6925319
#rf         0.4157637 0.5671968 0.3813019 0.4517406 0.7723533 0.6925319 1.0000000

# stack using glm
stackControl <- trainControl(method="repeatedcv", number=10, repeats=3,
savePredictions=TRUE, classProbs=TRUE)
set.seed(seed)
stack.glm <- caretStack(models, method="glm", metric="Accuracy", trControl=stackControl)
print(stack.glm)


# stack using random forest
set.seed(seed)
stack.rf <- caretStack(models, method="rf", metric="Accuracy", trControl=stackControl)
print(stack.rf)

res <- predict(stack.rf, data_test_kaggle)
toSave <- res
toSave<-as.data.frame(toSave)
toSave$toSave<-unfactor(toSave$toSave)
for(i in 1:250){
  if(toSave[i,1]=="alive"){
    toSave[i,1]="1"
  }else{
    toSave[i,1]="0"
  }
}


write.table(toSave, sep=",")
```